Capítulo 1

Algoritmos

La palabra algoritmo aparecerá gran cantidad de veces a lo largo de todo este libro. Corresponde, entonces, que dediquemos un capítulo a aclarar bien su significado. Estableceremos claramente qué es un algoritmo, cómo se especifica, cómo se implementa y cómo se mide su eficiencia. Además, veremos de qué manera, a lo largo de la historia, los algoritmos pasaron del dominio de las matemáticas al de las ciencias de la computación.

¿Qué son los algoritmos?	**16**
La máquina de Turing	17
Especificación de algoritmos	**18**
Implementación de algoritmos	**20**
Eficiencia de los algoritmos	21
Clases de algoritmos	**21**
Los algoritmos en la historia	**24**
Resumen	**25**
Actividades	**26**

¿QUÉ SON LOS ALGORITMOS?

Un algoritmo es un conjunto finito de instrucciones precisas que realizan una tarea, la cual, dado un estado inicial, culminará por arrojar un estado final reconocible.

Esta definición asume que la ejecución del algoritmo concluye en algún momento, dejando fuera los procedimientos que ejecutan permanentemente sin detenerse. Para incluir a éstos en la definición, algunos autores prefieren obviar la condición de que la ejecución concluya, con lo cual basta con que un procedimiento sea una secuencia de pasos que puede ser ejecutada por una entidad para que se lo considere algoritmo.

En el caso que no haya un estado final reconocible, el éxito del algoritmo no puede definirse como la culminación del proceso con un resultado significativo. En cambio, se requiere una definición de éxito que contemple secuencias ilimitadas de resultados, por ejemplo, un sistema de compresión/descompresión de datos en tiempo real (como los utilizados en el manejo de voz sobre IP); en este caso, el algoritmo no define por sí mismo la finalización del proceso, debiendo seguir su funcionamiento mientras haya datos para procesar. El éxito del algoritmo estará dado por el hecho de que los datos, una vez descomprimidos, sean iguales que antes de comprimirse.

El concepto de algoritmo se ilustra frecuentemente comparándolo con una receta: al igual que las recetas, los algoritmos habitualmente están formados por secuencias de instrucciones que probablemente se repiten (iteran) o que requieren decisiones (comparaciones lógicas) hasta que completan su tarea. Un algoritmo puede no ser correcto, con lo cual, por más que sus pasos se lleven a cabo correctamente, el estado final no será el esperado.

Normalmente, cuando un algoritmo está asociado con el procesamiento de información, se leen datos de una fuente o dispositivo de entrada, se procesan y se emiten por un dispositivo de salida, o bien se almacenan para su uso posterior. Los datos almacenados se consideran parte del estado interno de la entidad que ejecuta el algoritmo.

III VOZ SOBRE IP

La tecnología de voz sobre IP se utiliza para realizar comunicaciones telefónicas sobre redes IP (Internet Protocol). En esta tecnología son cruciales los algoritmos de compresión de datos, ya que, cuanto más se compriman los datos que representan la voz digitalizada, mejor será la calidad de comunicación.

Dado que un algoritmo es una lista precisa de pasos, el orden de ejecución será casi siempre crítico para su funcionamiento. En general, se asume que las instrucciones se enumeran explícitamente, y deben ejecutarse "desde arriba hacia abajo", lo cual se establece más formalmente según el concepto de **flujo de control**.

Esta forma de "pensar" el algoritmo asume las premisas del paradigma de programación imperativa. Dicho paradigma es el más común, e intenta describir las tareas en términos "mecánicos" y discretos. Los paradigmas de la programación funcional y de la programación lógica describen el concepto de algoritmo en una forma ligeramente diferente (en el **Capítulo 2** se detallan los distintos tipos de paradigmas).

Hasta aquí hemos dado una definición ciertamente informal del concepto de algoritmo. Para definirlo en forma matemáticamente precisa, Alan Mathison Turing –famoso matemático inglés (1912-1954), cuyas contribuciones en el campo de la matemática y de la teoría de la computación le han valido ser considerado uno de los padres de la computación digital– ideó un dispositivo imaginario al que denominó **máquina de computación lógica** (LCM, *Logical Computing Machine*), pero que ha recibido en su honor el nombre de **máquina de Turing**. Lo que confiere a este supuesto dispositivo su extraordinaria importancia es que es capaz de resolver cualquier problema matemático, a condición de que el mismo haya sido reducido a un algoritmo. Por este motivo, se considera que algoritmo es cualquier conjunto de operaciones que pueda ser ejecutado por la máquina de Turing (o, lo que es lo mismo, por un sistema Turing completo, tal como se explica más adelante).

La máquina de Turing

Una máquina de Turing es un autómata que se mueve sobre una secuencia lineal de datos. En cada instante, la máquina puede leer un único dato de la secuencia (generalmente un carácter) y realizar ciertas acciones en base a una tabla que tiene en cuenta su estado actual (interno) y el último dato leído. Entre las acciones que puede realizar, está la posibilidad de escribir nuevos datos en la secuencia, recorrer la secuencia en ambos sentidos y cambiar de estado dentro de un conjunto finito de estados posibles.

III FLUJO DE CONTROL

El flujo de control es el orden en que se ejecutan las instrucciones de un programa. Normalmente, el flujo de control de un programa es secuencial –es decir, las instrucciones se ejecutan una detrás de otra desde la primera hasta la última–, a menos que existan bifurcaciones o ejecución paralela.

Un sistema **Turing completo** es aquel que puede simular el comportamiento de una máquina de Turing. Dejando de lado las limitaciones impuestas por la capacidad de almacenamiento o la memoria, las computadoras actuales y los lenguajes de programación de propósito general definen sistemas Turing completos.

Independientemente de su forma concreta, cualquier dispositivo que se comporte como un sistema Turing completo puede en principio ejecutar cualquier cálculo que realice cualquier computadora; lógicamente, esta afirmación no considera la posible dificultad de escribir el programa correspondiente o el tiempo que pueda requerir realizar el cálculo en cuestión.

Podemos consultar la sección de **Servicios al lector** para conocer los sitios en donde obtener más información acerca de los sistemas Turing completos.

ESPECIFICACIÓN DE ALGORITMOS

Para cualquier proceso computacional, el algoritmo correspondiente debe estar rigurosamente definido, es decir, debe especificarse la forma en que se aplica a cada posible circunstancia que pueda surgir. Todos los casos deben estar contemplados, y el criterio que determina cada uno de ellos debe ser claro y computable.

En general, no existe un único algoritmo para cada problema que se quiere resolver. Diferentes algoritmos pueden completar la misma tarea, requiriendo cada uno diferentes cantidades de tiempo, espacio o esfuerzo. Sin embargo, la especificación puede ser exactamente la misma para todos ellos.

Para especificar un algoritmo de forma tal que su implementación sea correcta –es decir, que haga exactamente lo que se espera de él– y que, a la vez, pueda implementarse con diferentes lenguajes o herramientas, un método consiste en definir sus entradas y salidas, con sus correspondientes **precondiciones** y **poscondiciones**.

{} TURING TARPIT

Existe una categoría de lenguajes de programación, denominada "Turing Tarpit" (tarpit = pozo de alquitrán), la cual engloba a diversos lenguajes que cumplen la condición de ser Turing-completos, pero imponen restricciones extremas que los hacen difíciles de usar, aunque interesantes de analizar. Encontraremos más información sobre estos lenguajes en el **Apéndice D**.

A modo de ejemplo, veamos la especificación de un algoritmo que busca el máximo número en una lista:

Algoritmo: BuscarMaximo

- **Datos de entrada:** una lista *l* de *n* elementos numéricos.
- **Datos de salida:** un número *m*.
- **Precondiciones:**
  n es un número natural.
  n es mayor que cero (o sea, la lista no puede estar vacía).
  todos los elementos de *l* son números racionales.
- **Poscondiciones:**
  m es un número racional.
  m es el mayor de los elementos de *l*.

Esta especificación define de manera inequívoca cómo debe funcionar nuestro algoritmo. Sin embargo, por estar expresado en lenguaje natural –con toda su carga de ambigüedades–, puede prestarse a confusiones (quizá no en este caso, porque es un algoritmo muy simple, pero sí para casos más complejos). Por ese motivo, conviene expresar las especificaciones en un lenguaje más riguroso, como por ejemplo, las expresiones matemáticas usadas en el cálculo de predicados lógicos. Con tales consideraciones, podemos expresar nuestro algoritmo en forma más precisa:

Algoritmo: BuscarMaximo

- **Datos de entrada:** $l1 \ldots ln$
- **Datos de salida:** m
- **Precondiciones:**
  $n \in \mathbb{N}$
  $n > 0$
  $l_i \in \mathbb{Q} \; \forall \; 1 \leq i \leq n$
- **Poscondiciones:**
  $m \in \mathbb{Q}$
  $m \geq l_i \; \forall \; 1 \leq i \leq n$

No cabe duda de que esta especificación es más rigurosa, aunque seguramente es más difícil de entender para quien no domina esta clase de expresiones matemáticas.

En el **Capítulo 3** se analizan con mayor nivel de detalle las definiciones de precondiciones y poscondiciones de algoritmos, aplicadas a la especificación de tipos abstractos de datos (**TADs**).

IMPLEMENTACIÓN DE ALGORITMOS

La implementación es el proceso que toma la especificación del algoritmo y la traduce a una forma que pueda aplicarse a la solución del problema para el cual fue diseñado.

La implementación puede tomar formas muy diversas: podría significar la construcción de un circuito eléctrico o de un dispositivo mecánico que cumpla con las condiciones especificadas. Pero restrinjamos la definición al campo de la informática: en este sentido, implementar significa traducir el algoritmo a un lenguaje que pueda ser interpretado por un motor de ejecución.

Para el análisis y estudio de los algoritmos usualmente se utiliza una forma abstracta de implementación, la cual no utiliza un lenguaje de programación específico, sino que emplea formas de representar el algoritmo que luego pueden ser directamente traducidas a un lenguaje en particular. Algunas de estas formas son los **diagramas de flujo**, los **diagramas de bloques** y el **seudo código**. Este último es "casi" un lenguaje imperativo, con la salvedad de que no toma en cuenta los tipos de datos y, además, sus instrucciones pueden estar en idioma español o en cualquier otro, ya que no serán interpretadas por ninguna computadora.

Un sencillo ejemplo de la implementación en seudo código de nuestro algoritmo **BuscarMaximo** sería la siguiente:

```
Función BuscarMaximo(lista)
    Mayor = lista(1)
    Contador = 2
    Mientras Contador ≤ longitud(lista) hacer
        Si lista(Contador) > Mayor entonces
            Mayor = lista(Contador)
        Fin Si
        Contador = Contador + 1
```

III PRECONDICIONES Y POSCONDICIONES

Las precondiciones son las condiciones que el algoritmo asume que deben cumplir los datos de entrada. Las poscondiciones describen cómo deben estar calculados los datos de salida.

```
    Fin Mientras
    Devolver Mayor
Fin Función
```

El seudo código tiene la ventaja de poder derivarse con poco esfuerzo en casi cualquier lenguaje imperativo, como el **Pascal** o el **C**.

Eficiencia de los algoritmos

La especificación de un algoritmo puede incluir consideraciones sobre su eficiencia, dado que una implementación incorrecta puede hacer que demore en ejecutarse mucho más tiempo de lo aceptable. Para ello se utilizan notaciones que expresan la **complejidad** de los algoritmos en función del volumen de datos a procesar (ver el **Capítulo 4** para mayor información). Una de estas notaciones es la denominada "la gran O", que indica la cantidad de veces que el algoritmo debe repetir su bloque principal de instrucciones para hacer su trabajo.

El ciclo principal del algoritmo **BuscarMaximo** –explicado en la página 20– recorre una vez toda la lista de *n* elementos para determinar cuál es el mayor. Su bloque principal (el delimitado entre **Mientras** y **Fin Mientras**) se repite tantas veces como elementos haya en la lista. Por lo tanto, se dice que su complejidad es **O(n)** o que tiene complejidad lineal, ya que el tiempo que demora en ejecutarse el algoritmo aumentará proporcionalmente a la cantidad de elementos que tenga la lista.

CLASES DE ALGORITMOS

Una forma de clasificar los algoritmos consiste en diferenciarlos por su metodología de diseño. A continuación se presenta una síntesis de las metodologías más comunes, aplicables cada una a diferentes clases de problemas:

{ } BASIC Y PASCAL

Obsérvese que, si se cuenta con un algoritmo escrito en seudo código y se lo traduce al inglés, prácticamente se tendrá el mismo algoritmo escrito en un lenguaje "cercano" al Pascal o al Basic.

III COMPLEJIDAD

Cuando se habla de complejidad de algoritmos, no se hace referencia a lo fácil o difícil que puede ser entenderlos, sino a cuánto trabajo llevará su ejecución.

- **Fuerza bruta**: los algoritmos de fuerza bruta resuelven el problema con la estrategia más obvia de solución, que no siempre es la mejor según el número de operaciones que se requiere.

- *Divide and conquer* **(divide y reinarás)**: esta metodología divide las instancias del problema a resolver en instancias cada vez más pequeñas, usualmente en forma recursiva, hasta llegar a una instancia en que el problema es resoluble en forma trivial o con unas pocas instrucciones. Los algoritmos de búsqueda binaria son un ejemplo de la metodología *divide and conquer* (**Figura 1**).

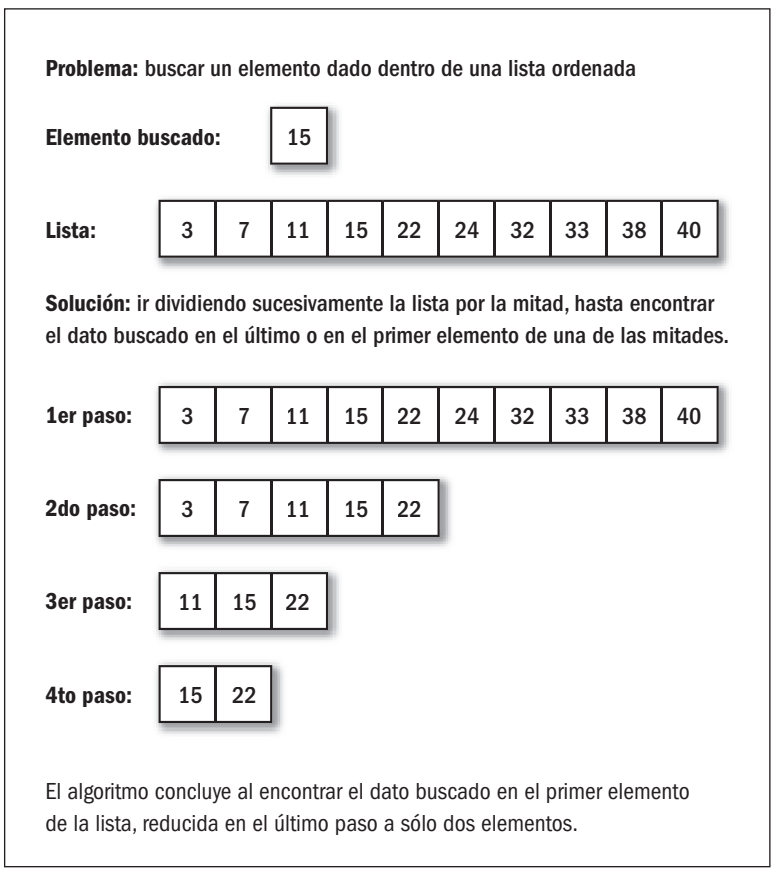

Problema: buscar un elemento dado dentro de una lista ordenada
Elemento buscado:
15
Lista:
3 7 11 15 22 24 32 33 38 40
Solución: ir dividiendo sucesivamente la lista por la mitad, hasta encontrar el dato buscado en el último o en el primer elemento de una de las mitades.
1er paso:
3 7 11 15 22 24 32 33 38 40
2do paso:
3 7 11 15 22
3er paso:
11 15 22
4to paso:
15 22
El algoritmo concluye al encontrar el dato buscado en el primer elemento de la lista, reducida en el último paso a sólo dos elementos.

Figura 1. *La metodología* ***divide and conquer*** *divide el problema en instancias cada vez más pequeñas, hasta llegar a una en que la solución es trivial.*

- **Programación dinámica**: cuando un problema presenta una subestructura óptima –o sea, cuando la solución óptima de un problema se obtiene a partir de las soluciones óptimas de sus subproblemas–, se encuentra la solución resolviendo primero los subproblemas más sencillos y luego utilizando esas subsoluciones para resolver problemas incrementalmente difíciles. Por ejemplo, si se tiene una se-

rie de puntos (definidos por coordenadas x, y) que delimitan una región, y se necesita saber si otro punto se encuentra dentro o fuera de esa región, una forma de resolver el problema consiste en comenzar formando cuadrados con puntos contiguos, para luego formar figuras cada vez más grandes y, por cada figura, determinar si el punto está dentro o fuera de ella (**Figura 2**). En cada paso se aprovecha la información de los pasos anteriores, hasta que, al completar todos los puntos, se obtiene el resultado del problema.

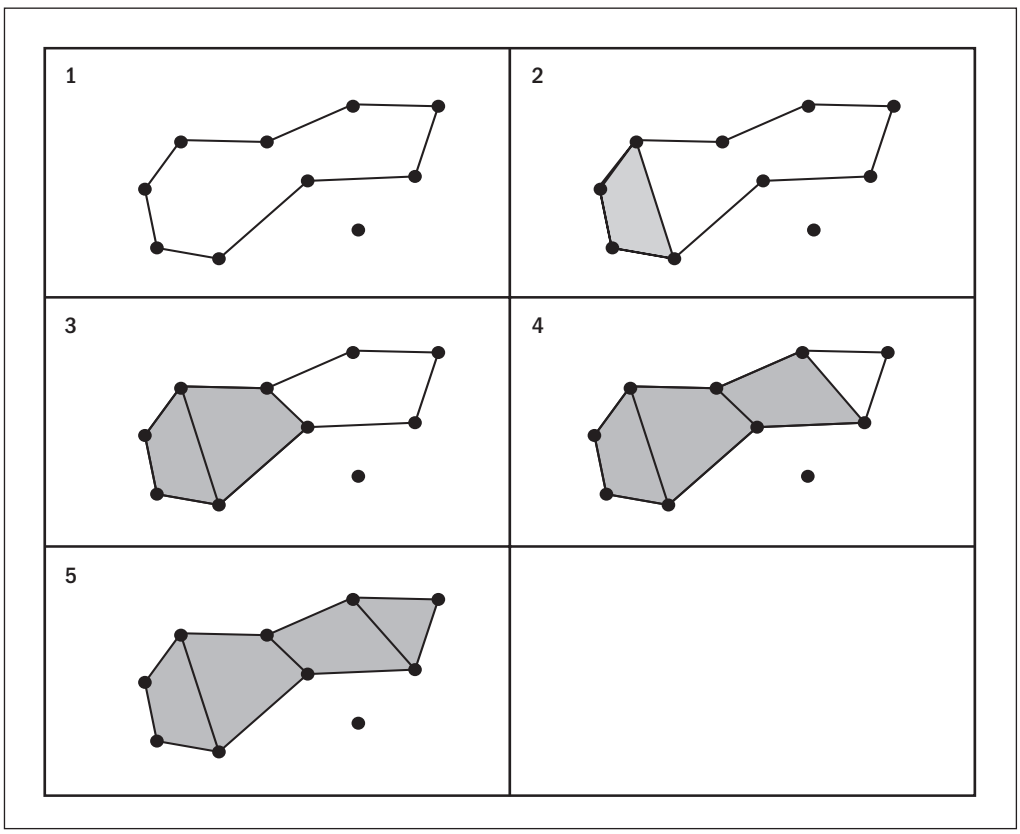


***Figura 2.** El problema de determinar si un punto está dentro o fuera del área delimitada por una serie de otros puntos se resuelve naturalmente por programación dinámica.*

III GRAFO

Un grafo es un objeto matemático utilizado para modelar problemas que involucran circuitos, redes y, en general, toda aquella entidad que pueda representarse por medio de nodos (vértices) y líneas que los conectan (aristas).

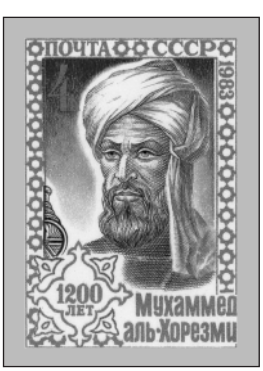
- **Programación lineal**: para resolver un problema utilizando programación lineal, se plantea una serie de inecuaciones y luego se busca maximizar (o minimizar) las variables, respetando las inecuaciones. Muchos problemas pueden plantearse en la forma de un conjunto de inecuaciones, para luego resolverlos utilizando un algoritmo genérico, como por ejemplo, el denominado método **Simplex** (en la sección de **Servicios al lector** se detallan sitios donde obtener más información sobre este método).

- **Búsqueda y enumeración**: muchos problemas (como por ejemplo, un juego de ajedrez) pueden modelarse con grafos y resolverse a partir de un algoritmo de exploración del grafo. Tal algoritmo especificará las reglas para moverse en el grafo en busca de la solución al problema. Esta categoría incluye también algoritmos de *backtracking* (vuelta atrás), los cuales van ensayando distintos caminos con posibles soluciones y vuelven atrás cuando no las encuentran. Por ejemplo, para encontrar la salida en un laberinto, cada vez que se llega al final de un camino se vuelve atrás hasta la última bifurcación, para probar con las distintas alternativas de esa bifurcación.

- **Algoritmos heurísticos**: el propósito de estos algoritmos no es necesariamente encontrar una solución final al problema, sino encontrar una solución aproximada cuando el tiempo o los recursos necesarios para encontrar la solución perfecta son excesivos. Muchos sistemas antivirus utilizan métodos heurísticos para detectar conductas de programas que podrían estar actuando en forma maliciosa.

- **Algoritmos voraces**: seleccionan la opción de solución (solución local) que tenga un costo menor en la etapa de solución en la que se encuentran, sin considerar si esa opción es parte de una solución óptima para el problema completo (solución global).

LOS ALGORITMOS EN LA HISTORIA



El origen del término "algoritmo" se remonta al siglo IX y se le atribuye su invención al matemático árabe **Abu Ja'far Muhammad ibn Musa al-Khwarizmi** (**Figura 3**).

Figura 3. *Abu Ja'far Muhammad ibn Musa al-Khwarizmi, padre de los algoritmos. La imagen corresponde a una estampilla de la ex Unión Soviética.*

La palabra algoritmo se refería originalmente sólo a las reglas de la aritmética con números arábigos. Recién en el siglo XVIII se expandió su significado para abarcar en su definición a toda clase de procedimientos utilizados con el propósito de resolver problemas o realizar determinadas tareas.

El primer caso de una algoritmo escrito para una computadora se considera que son las notas escritas por **Ada Byron** en 1842 para el motor analítico de **Charles Babbage**. Por esta razón, se considera a Ada Byron como la primera programadora de la historia. Sin embargo, dado que Babbage nunca terminó su motor analítico, el algoritmo jamás llegó a implementarse.

RESUMEN

En este capítulo hemos analizado en detalle uno de los elementos básicos de los que trata este libro: el algoritmo. Comenzamos por su definición –tanto informal, comparándolo con una receta, como formal, validándolo con la máquina de Turing– y seguimos por la forma de especificarlo, implementarlo y clasificarlo en categorías. Finalmente, descubrimos que el primer programador de la historia fue una mujer.

✔ ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Qué es un algoritmo?

2 ¿Para qué sirve la tarea de especificar un algoritmo?

3 ¿Cuál será la metodología más conveniente para diseñar un algoritmo que juegue al famoso ta-te-ti?

4 ¿Cuál sería un caso práctico para que resulte óptimo aplicar la técnica de programación lineal?

EJERCICIOS PRÁCTICOS

✔ Construya un programa que implemente una máquina de Turing. El programa debe mantener un vector de datos y debe ser capaz de interpretar una secuencia de instrucciones que manipulen los datos del vector, de a uno por vez.

✔ Escriba la especificación e implementación (en seudo código o en cualquier lenguaje que prefiera) de un algoritmo que obtenga el mínimo elemento de una lista de valores numéricos.

✔ Escriba la especificación –poniendo especial atención a las poscondiciones– de un algoritmo que ordena un vector de n elementos.